Multiarchitecture Programming for Accelerated Compute, Freedom of Choice for Hardware

Intel® oneAPI HPC Toolkit

Programming for Distributed HPC systems using Intel® MPI Library

Intel® oneAPI Tools for HPC

Intel® oneAPI HPC Toolkit

Deliver Fast Applications that Scale

What is it?

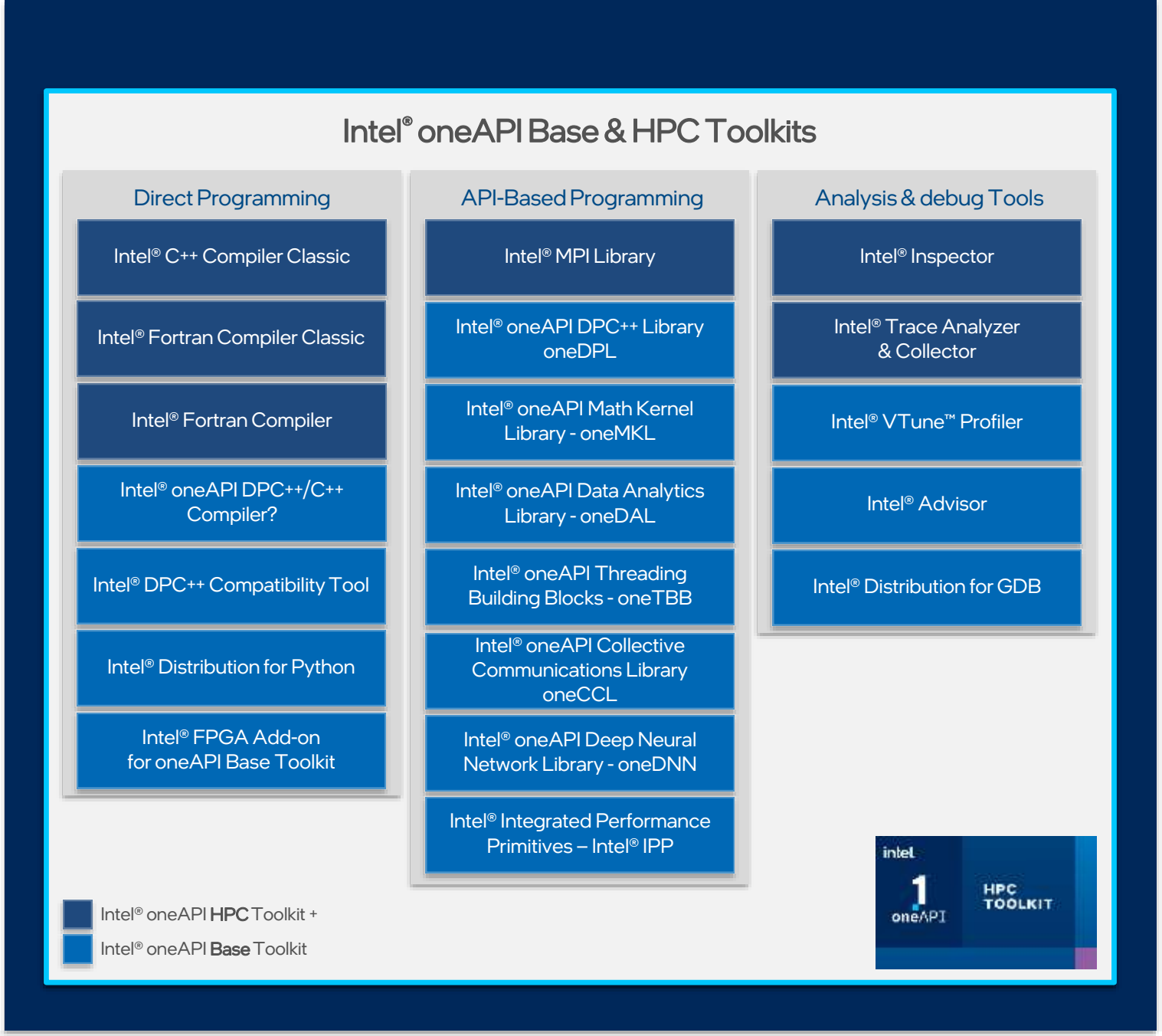
A toolkit that adds to the Intel® oneAPI Base Toolkit for building high-performance, scalable parallel code on C++, Fortran, SYCL, OpenMP & MPI from enterprise to cloud, and HPC to AI applications.

Who needs this product?

- OEMs/ISVs
- C++, Fortran, OpenMP, MPI Developers

Why is this important?

- Accelerate performance on Intel® Xeon® & Core™ processors & Intel accelerators
- Deliver fast, scalable, reliable parallel code with less effort built on industry standards



Intel® MPI Library

Key Features:

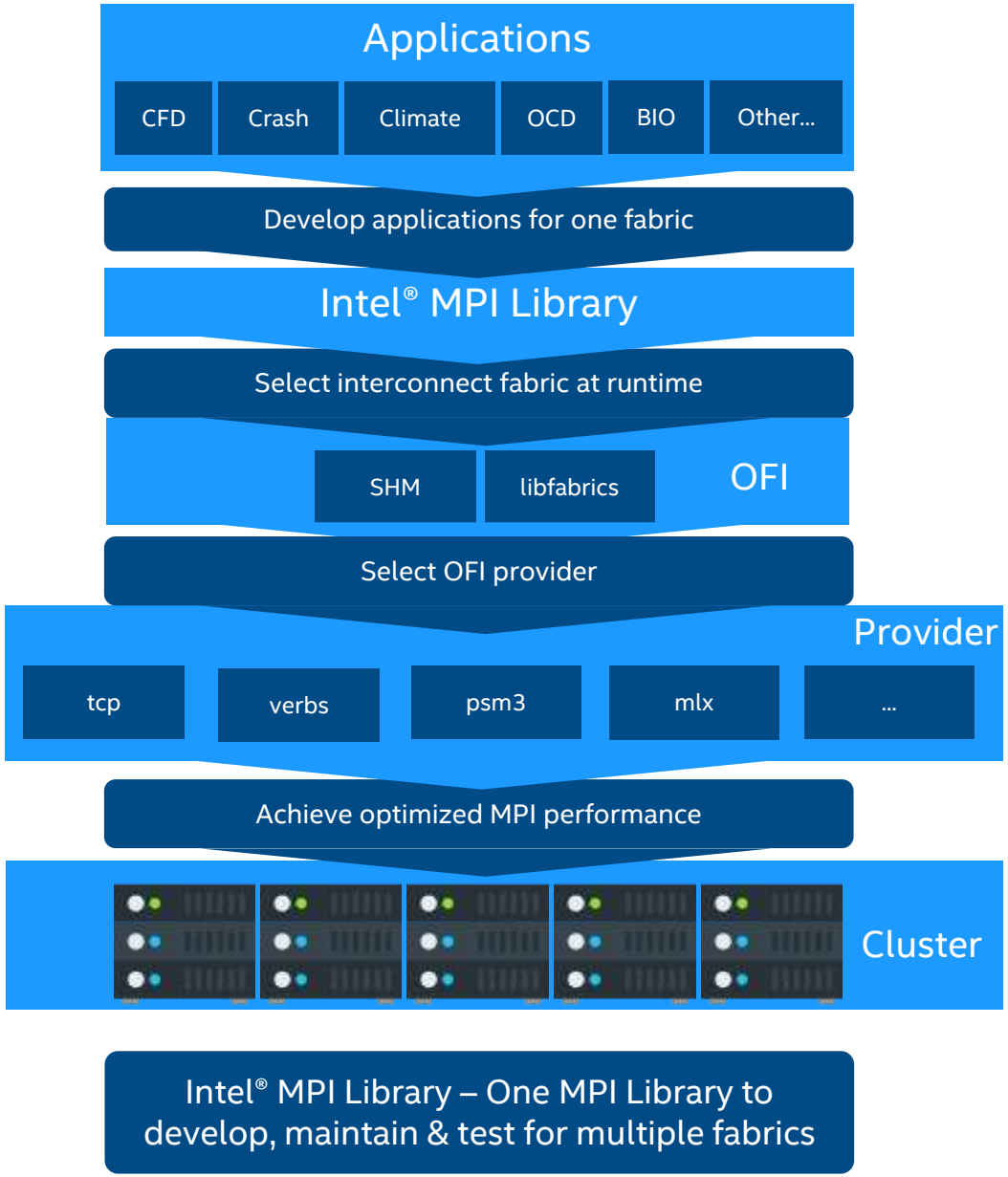
- MPI-1, MPI-2.2 and MPI-3.1 specification conformance
- Interconnect independence
- C, C++, Fortran 77, 90 & 2008 language bindings
- Amazon* AWS/EFA, Google* GCP support
- Intel GPU pinning & buffers support

2021.8 Release notes:

- initial support for the Intel® Data Center GPU MAX Series (formerly code-named Ponte Vecchio) utilizing XE Link for direct GPU to GPU communications
- enabled the new embedded Data Streaming Accelerator in 4th Generation Xeon Scalable Processors (formerly code-named Sapphire Rapids).

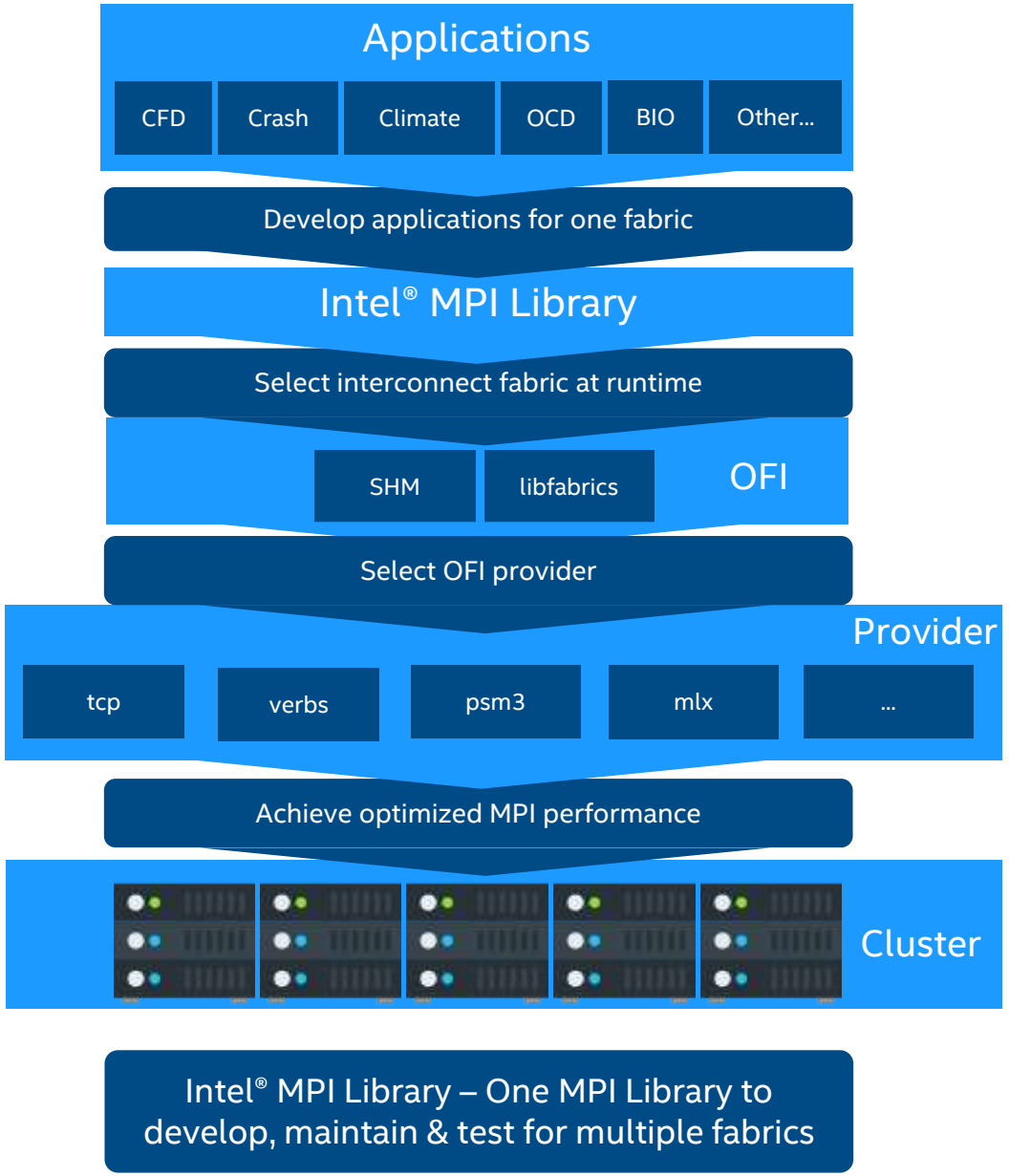
2021.10 Release notes (latest):

- …
- Large counts support for ILP64 (point-to-point, collectives)
- Waitmode support (Tech preview)



Selecting Fabrics & Providers

- Runtime selection via environment variables:
- I_MPI_FABRICS
  - shm: optimized for shared-memory; can only be used if all ranks are intranode
  - ofi: general-purpose fabric; requires a provider
- FI_PROVIDER
  - mlx: provider running over UCX
  - tcp: general purpose, over ethernet (e.g. for cloud applications)
  - psm3: Intel's newest provider, strongly recommended for GPU features



See more: https://www.intel.com/content/www/us/en/docs/mpi-library/developer-guide-linux/2021-6/ofi-providers-support.html

Intel® MPI Simplifies Programming for GPUs*

- simple distribution with one MPI rank per tile/GPU
- uncomplicated pinning via environment variables
- transparent communication with GPU buffers – no host-copy necessary!

Intel® MPI - GPU Pinning

See more: https://www.intel.com/content/www/us/en/docs/mpi-library/developer-reference-linux/2021-8/gpu-pinning.html

Intel® GPU Pinning Support Overview

- Automatic Intel GPU resources distribution
- No user code changes required on different GPU configurations
- NUMA aware

Default settings:

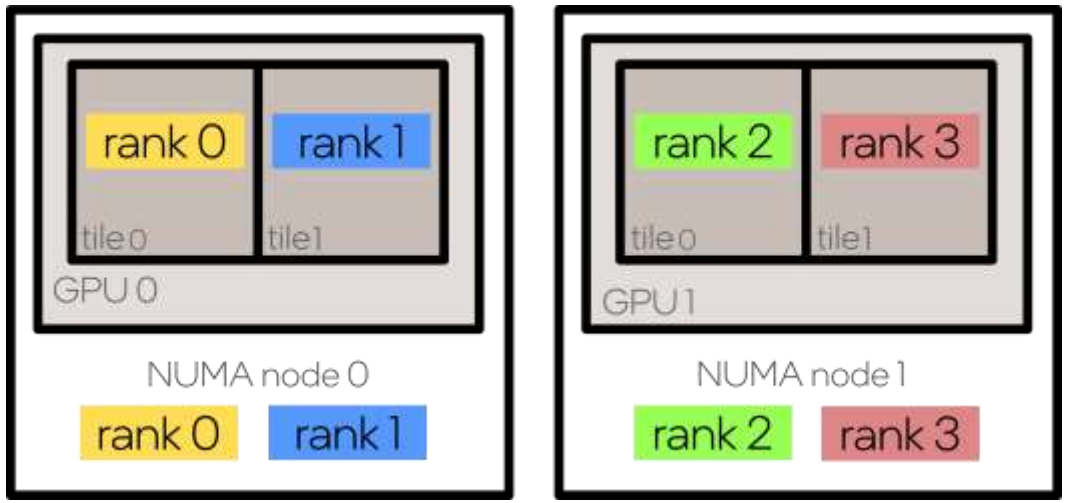
I_MPI_OFFLOAD_* family:

TOPOLIB=level_zero
CELL=tile
DOMAIN_SIZE=-1 (auto)
DEVICES=all

E.g: I_MPI_OFFLOAD_TOPOLIB=level_zero

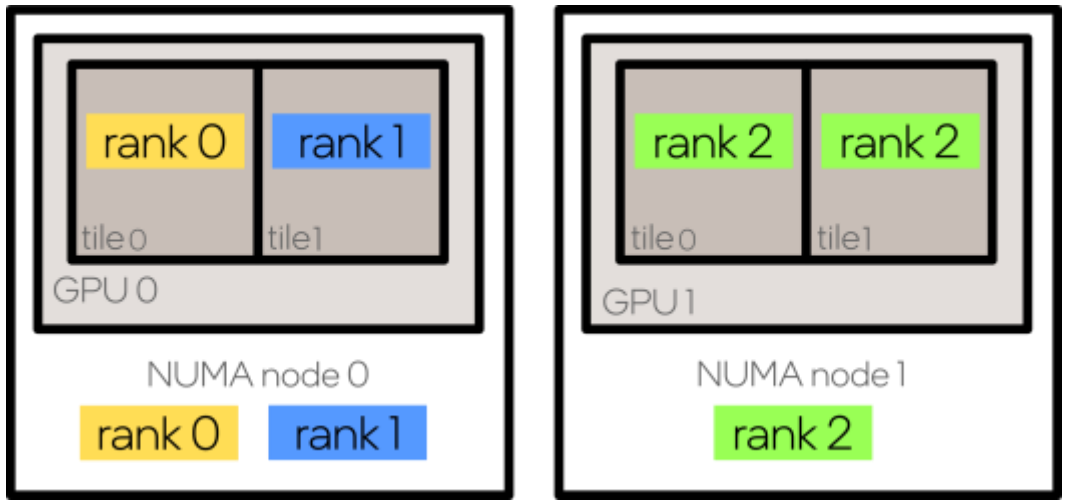
Example – Default: 4 ranks

```
I_MPI_DEBUG=120
[0] MPI startup(): ===== GPU topology on host1 =====
[0] MPI startup():   NUMA nodes  : 2
[0] MPI startup():   GPUs        : 2
[0] MPI startup():   Tiles       : 4
[0] MPI startup():
=====  GPU Placement on packages  =====
[0] MPI startup():  NUMA Id     GPU Id   Tiles       Ranks
[0] MPI startup():  0           0        (0,1)       0,1
[0] MPI startup():  1           1        (2,3)       2,3
```



Example – Default: 3 ranks

```
I_MPI_DEBUG=120
[0] MPI startup(): ===== GPU topology on host1 =====
[0] MPI startup():   NUMA nodes  : 2
[0] MPI startup():   GPUs        : 2
[0] MPI startup():   Tiles       : 4
[0] MPI startup():
=====  GPU Placement on packages  =====
[0] MPI startup():  NUMA Id     GPU Id   Tiles       Ranks
[0] MPI startup():  0           0        (0,1)       0,1
[0] MPI startup():  1           1        (2,3)       2
```



Example – I_MPI_OFFLOAD_DOMAIN_SIZE

I_MPI_OFFLOAD_DOMAIN_SIZE=1

I_MPI_DEBUG=3

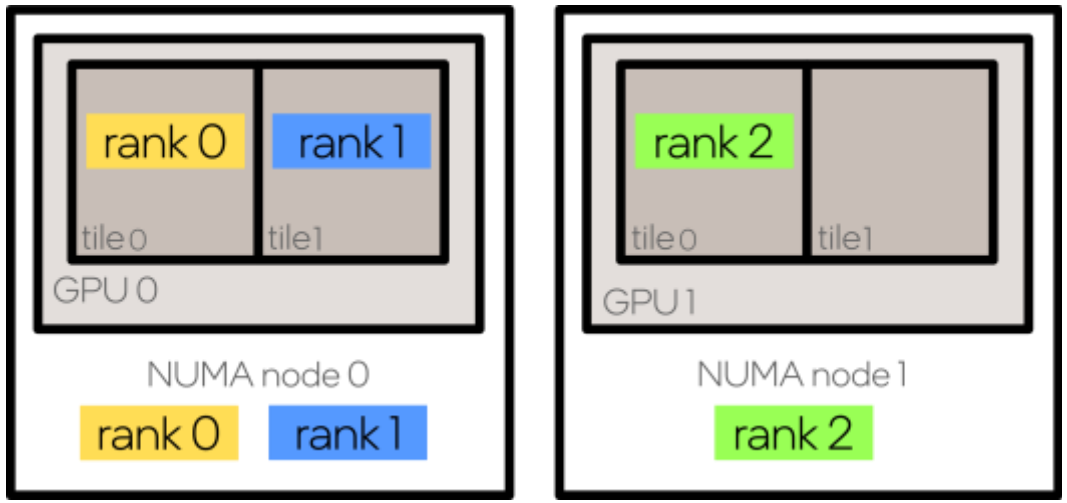
[0] MPI startup(): ===== GPU pinning on host1 =====

[0] MPI startup(): Rank Pin tile

[0] MPI startup(): 0 {0}

[0] MPI startup(): 1 {1}

[0] MPI startup(): 2 {2}



Example – I_MPI_OFFLOAD_CELL

I_MPI_OFFLOAD_CELL=device

I_MPI_DEBUG=3

[0] MPI startup(): ===== GPU pinning on host1 =====

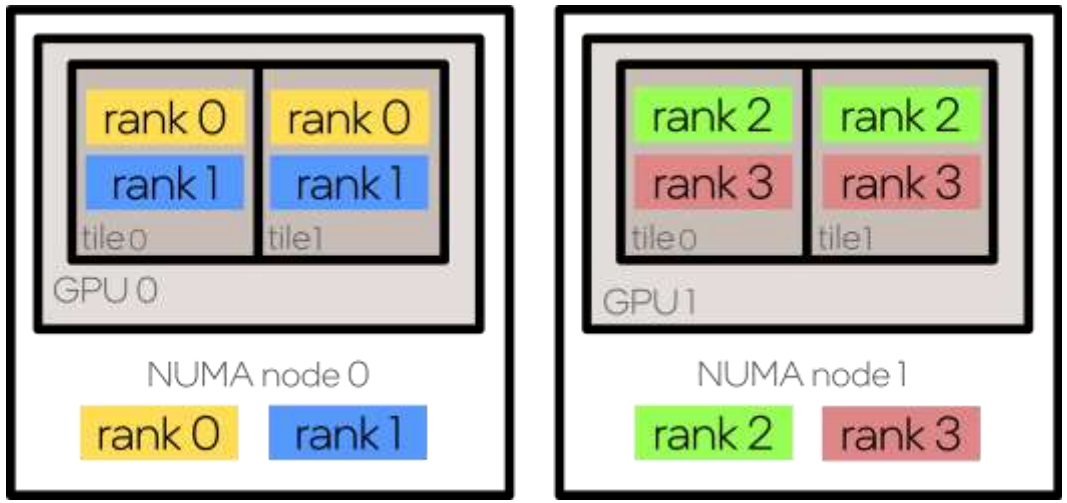
[0] MPI startup(): Rank Pin tile

[0] MPI startup(): 0 {0,1}

[0] MPI startup(): 1 {0,1}

[0] MPI startup(): 2 {2,3}

[0] MPI startup(): 3 {2,3}



Example – I_MPI_OFFLOAD_DEVICES

I_MPI_OFFLOAD_DEVICES=0

I_MPI_DEBUG=3

[0] MPI startup(): ===== GPU pinning on host1 =====

[0] MPI startup(): Rank Pin tile

[0] MPI startup(): 0 {0}

[0] MPI startup(): 1 {1}

[0] MPI startup(): 2 {0}

[0] MPI startup(): 3 {1}

Example – I_MPI_OFFLOAD_DOMAIN

I_MPI_OFFLOAD_DOMAIN=[B,2,5,C]

reverse bit masks: [1101,0100,1010,0011]

I_MPI_DEBUG=3

[0] MPI startup(): ===== GPU pinning on host1 =====

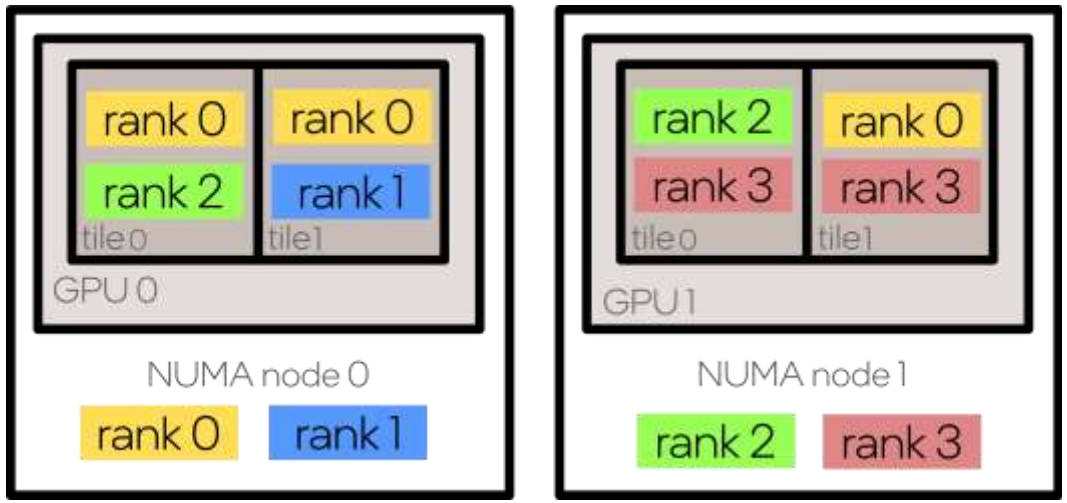
[0] MPI startup(): Rank Pin tile

[0] MPI startup(): 0 {0,1,3}

[0] MPI startup(): 1 {1}

[0] MPI startup(): 2 {0,2}

[0] MPI startup(): 3 {2,3}



Intel® MPI - GPU Buffers Support

See more: https://www.intel.com/content/www/us/en/docs/mpi-library/developer-reference-linux/2021-8/gpu-buffers-support.html

Execution models – OpenMP Offloading

1 - Old copy-back technique

```
#pragma omp target data map(to: ..., b[0:len]) map(tofrom: c[0:len])
{
  double *a = (double*) omp_target_alloc(len * sizeof(double), 0);

  for (int j=0; j < nrep; j++) {
    // Compute on GPU
    #pragma omp target teams distribute parallel for
    for (int i = 0; i < len; i++) {
      a[i] = sin(b[i] + alpha * c[i]);
    }

}
// Copy data back to host and send from there
MPI_Sendrecv(c, len, MPI_DOUBLE, destRank,   1,
             b, len, MPI_DOUBLE, sourceRank, 1, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
```

Execution models – OpenMP Offloading

2 – GPU-buffer aware

```
#pragma omp target data map(to: ..., b[0:len]) map(tofrom: c[0:len])
                use_device_ptr(b,c)
{
  double *a = (double*) omp_target_alloc(len * sizeof(double), 0);

  for (int j=0; j < nrep; j++) {
    // Compute on GPU
    #pragma omp target teams distribute parallel for is_device_ptr(b,c)
    for (int i = 0; i < len; i++) {
      a[i] = sin(b[i] + alpha * c[i]);1
    }

    // GPU-aware MPI sendrecv
    MPI_Sendrecv(b, len, MPI_DOUBLE, destRank,   1,
                 c, len, MPI_DOUBLE, sourceRank, 1, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
}
```

For SYCL with USM: automatically GPU-

Execution models – OpenMP Offloading

3 – Compiling & Running

- C language:

```
$ mpiicc -cc=icx -fiopenmp -fopenmp-targets=spir64 test.c -o test
```

- Fortran language:

```
$ mpiifort -fc=ifx -fiopenmp -fopenmp-targets=spir64 test.f90 -o test
```

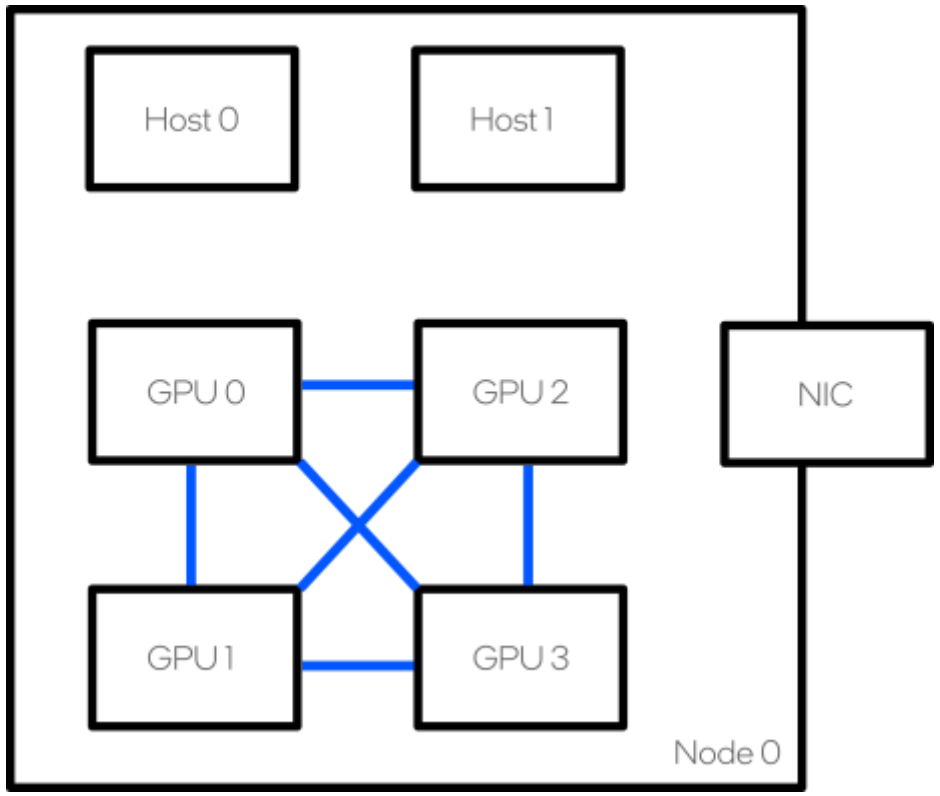
- Launch application as usual:

```
$ I_MPI_OFFLOAD=1 mpirun –n 8 test
```

GPU Buffer Control Variables

- I_MPI_OFFLOAD_IPC:

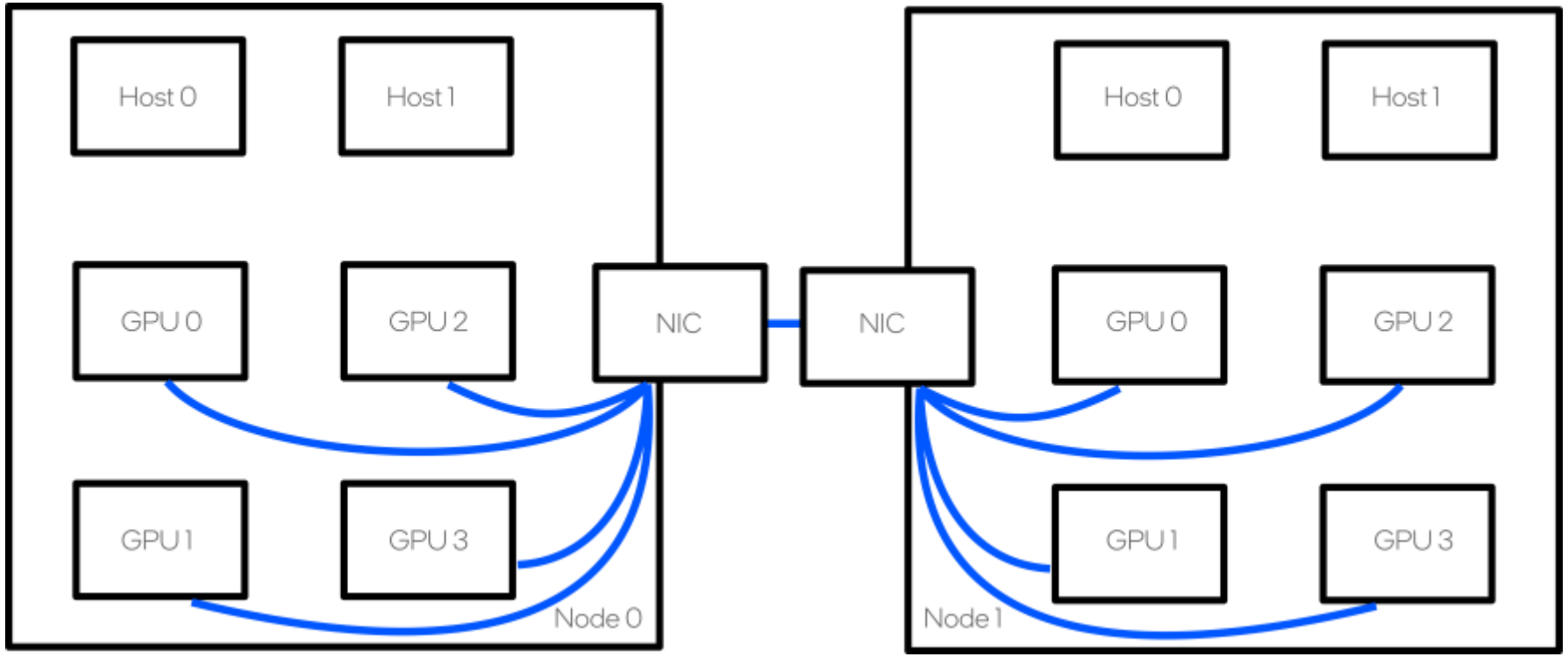
  toggles GPU IPC (i.e. XeLink; single-node)



See more: https://www.intel.com/content/www/us/en/docs/mpi-library/developer-reference-linux/2021-8/gpu-buffers-support.html

GPU Buffer Control Variables

- I_MPI_OFFLOAD_RDMA: toggles D2D copy through NIC (multi-node)



See more: https://www.intel.com/content/www/us/en/docs/mpi-library/developer-reference-linux/2021-8/gpu-buffers-support.html

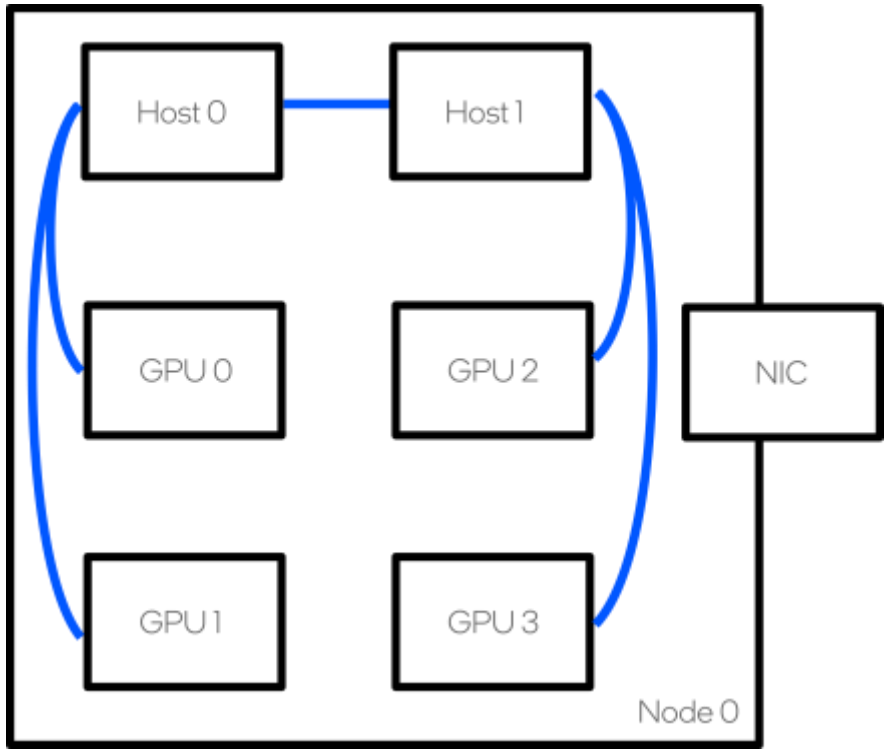
GPU Buffer Control Variables

- I_MPI_OFFLOAD_PIPELINE:

  toggles use of pipeline algorithms (for large message sizes)

- I_MPI_OFFLOAD_PIPELINE_THRESHOLD:

  uses pipeline for large messages (default is 65536)



See more: https://www.intel.com/content/www/us/en/docs/mpi-library/developer-reference-linux/2021-8/gpu-buffers-support.html

Tech Preview - IMB and IPC

See more: https://www.intel.com/content/www/us/en/docs/mpi-library/user-guide-benchmarks/2021-2/overview.html

Notices & Disclaimers

Performance varies by use, configuration and other factors. Learn more on the Performance Index site.

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See backup for configuration details. No product or component can be absolutely secure.

Your costs and results may vary.

Intel technologies may require enabled hardware, software or service activation.

Intel® MPI

Asynchronous Progress

- advanced feature
- enables non-blocking MPI calls to overlap communication and computation
- does not come "for free" - uses a progress thread at the background
- requires some fine-tuning
- **Runtime toggle** – no recompilation!

Key environment variables:

- I_MPI_ASYNC_PROGRESS
- I_MPI_ASYNC_PROGRESS_THREADS
- I_MPI_PROGRESS_PIN

See more: https://www.intel.com/content/www/us/en/docs/mpi-library/developer-guide-linux/2021-6/asynchronous-progress-control.html